(2)

Annual Report for Contract Number N00014-88-K-0641

For the period: 1 October 1988 - 30 September 1989

A. Nico Habermann
Carnegie Mellon University
School of Computer Science
(412) 268 - 2592
anh@cs.cmu.edu
Graduate Research on Miró and Avalon
Student Reporting - Mark Walter Maimone
N00014 - 88 - K - 0641
1 Oct 88 - 30 Sep 89

1. Miró Productivity Measures

Refereed papers submitted but not yet published: 1

Refereed papers published: 2

Unrefereed reports and articles: 3

Books or parts thereof submitted but not yet published: 1

Books or parts thereof published: 0

Patents filed but not yet granted: 0

Patents granted: 0

Invited presentations: 0

Honors received (fellowships, technical society appointments, conference committee role, editorship, etc): 0

Prizes or awards received: 0

Promotions obtained: 0

Graduate students supported >= 25% of full time: 1

Post-docs supported >= 25% of full time: 0

Minorities supported: 0

Statement A per telecon
Dr. Andre Van Tilborg ONR/Code 1133
Arlington, VA 22217-5000

NWW 4/10/92

A. Nico Habermann
Carnegie Mellon University
School of Computer Science
(412) 268 - 2592
anh@cs.cmu.edu
Graduate Research on Miró and Avalon
Student Reporting - Mark Walter Maimone
N00014 - 88 - K - 0641
1 Oct 88 - 30 Sep 89

2. Miró Summary of Technical Progress

The Miró group is designing and implementing a package of two visual languages for computer security specifications. The *instance* language describes static filesystem configurations: who are the users, and what files can they access at this moment in time? The *constraint* language defines sets of legal instance pictures: what configurations are allowed? In the past year, we have completed the design of these languages, and have begun to implement the tools necessary to make them part of a working software system.

A. Nico Habermann
Carnegie Mellon University
School of Computer Science
(412) 268 - 2592
anh@cs.cmu.edu
Graduate Research on Miró and Avalon
Student Reporting - Mark Walter Maimone
N00014 - 88 - K - 0641
1 Oct 88 - 30 Sep 89

3. Miró Detailed Summary of Technical Results

The Miró group is designing and implementing a package of two visual languages for computer security specifications. The *instance* language describes static filesystem configurations; who are the users, and what files can they access at this moment in time? The *constraint* language defines sets of legal instance pictures; what configurations are allowed? By October 1988, work on the instance language was essentially complete: a formal description of its semantics was presented at the Visual Language workshop that month in [1]. The constraint language then developed over the course of the 1988-1989 academic year. The rest of this summary presents highlights of the constraint language. A complete description can be found in [2].

3.1. Background

Miró is a visual language for specifying security configurations. By "visual language," we mean a language whose entities are graphical, such as boxes and arrows. By "specifying," we mean stating independently of any implementation the required and/or desired properties of a system. Finally, by "security," we mean security for operating systems: ensuring that files are protected from unauthorized access and granting privileges to perhaps some users, but not others.

Our work differs from other work in visual languages in three important ways: First, unlike many languages based on diagrams where boxes and lines may fail to have a precise meaning, or worse, have multiple interpretations, we are careful to provide a completely formal semantics for our visual language. Second, in contrast to visual programming languages, we are interested in specifications, not executable programs. Third, we do not use visualization just for the sake of drawing pretty pictures; instead, we address a domain, security, that lends itself naturally to a two-dimensional representation.

Security lends itself naturally to visualization because the domains of interest are best expressed in terms of relations on sets, easily depicted as Venn diagrams, and the connections among objects in these domains are best expressed as relations (e.g., access rights), easily depicted as edges in a graph (where the nodes are Venn diagrams). The underlying model is a two-dimensional access rights matrix, but the compact visual notation presents the same information in a more intuitive manner. The Miró instance and constraint languages extend Harel's work on higraphs, an elegant formalism that shows relations on Venn diagrams.

Figure 1 shows a Miró security specification that reflects some aspects of the Unix file protection scheme. The outermost left-hand box depicts a world, `World`, of users, two groups, `Group1`, and `Group2`, and two users, `Alice` and `Bob`. The containment and overlap relationships between the world, groups, and users indicate that all users are in the world, and that some users are members of more than one group. The right-hand box denotes the set of files in Alice's mail directory. The arrows indicate that Alice, and no other user, has read access to her mail files. She is granted read permission because the direct positive arrow from `Alice` overrides (i.e., is more tightly nested than) the negative arrow from `World`.

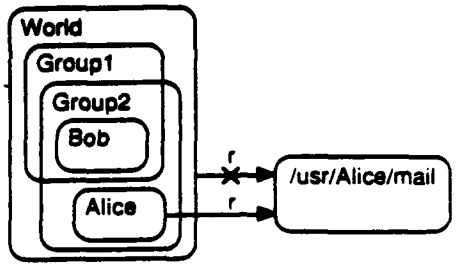


Figure 1: A Sample Miró Picture

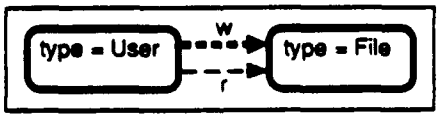
3.2. Constraint Language

Miró is expressive enough to specify security configurations for any operating system. However, the kinds of pictures users can draw will vary depending on the kind of system they are specifying. In particular, the system architecture will impose constraints on what should be considered a "legal" (realizable and acceptable) picture for that system. For example, a legal picture for Multics may be illegal for Unix.

Constraints themselves are specified by pictures drawn in a visual language similar to the Miró picture language outlined above. We make the distinction, therefore, between Miró *instance* pictures and Miró *constraint* pictures. Each constraint specifies a "pattern," which is a template for many different instance pictures. Instance pictures may *match* the pattern given by a constraint picture.

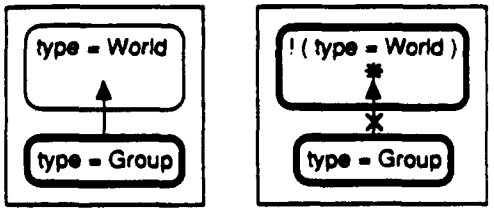
Constraints are typically statements in which the occurrence of some situation will imply that some further condition should hold. Therefore, constraints are divided into two parts: the antecedent (or *trigger*) and the consequent (or *requirement*). For example, we may wish to specify the constraint that any time a user has write access to a file, he should also have read access to it. In this case, the existence of write privilege is the "trigger" of the read privilege "requirement." Both parts are expressed together in a single constraint; the trigger is drawn with thick lines, the requirement with thin line. Following are two constraint picture examples:

1. Whenever a **User** has write access to a **File**, he should also have read access to that **File**.



---

2. Every **Group** must be directly contained in at least one **World**, and a **Group** can only be contained in a **World**.



The features of the constraint language are detailed in [2]; the following summarizes the constraint language constructs:

Syntax Arrows: These match against individual arrows in an instance picture.

Semantics Arrows: These match relations in the two-dimensional access matrix.

Containment Arrows: Containment arrows must join two boxes. They are matched when one instance box is *directly* contained within the other.

Starred Containment: Matches when one box is contained within the other *at any level* (i.e., there may be boxes surrounding the smaller, yet still contained within the larger).

Negated Arrows: Any of the arrows may be negated. For syntax arrows, it means there is a negative arrow joining two boxes: for semantic arrows, it means the relation is negative: for containment arrows, it means one box is not contained in the other.

Box Patterns: Each constraint box may have a predicate to restrict the kinds of instance boxes it matches. For example, *type = User & name = "jones"* would only match an instance User (as opposed to File) box with the name Jones.

Thick and Thin: For all parts of the instance picture matching the thick parts, there must exist instance subpictures matching the thin parts.

Numeric Constraints: A range limit can be given on the number of subpictures that match the thick part of the constraint.

3.3. Implementation

We are designing a set of tools that will allow us to exploit Miró's capabilities. Our front-end tools are operating system independent, and use an intermediate file format to represent instance pictures and constraints; the back-end tools use the intermediate file format to interface with actual operating systems.

The front-end tools include the *Miró graphical editor*, which allows users to view and modify instance pictures and constraints. The editor runs under the X Window system and is built on top of the Garnet system, developed at Carnegie Mellon. The editor provides simple syntactic checks, and translates pictures and constraints into our intermediate file format. It also provides the ability to "zoom" out and in to allow the user to abstract away or focus in on details of a picture, and to "highlight" the sub-pictures of interest. The *Miró printing package* takes the intermediate file format and produces a PostScript file of the Miró

picture. The *Miró static semantics checker* checks a picture for ambiguity or violations of constraints, and reports any errors.

The back-end tools include the *Miró file system checker*, which probes the file system to check whether a given file system's protection conforms with its instance language description. A different file system checker is needed for each operating system on which Miró will be used. We are investigating the feasibility of a *Miró file system inspection tool* which could alter an instance picture to correspond to the actual state of the file system. The file system inspection tool raises a number of interesting questions in the area of automated production of attractive graphs. Of the tools mentioned, we have prototypes of the graphical editor, static semantics checker, and the printing package.

In conclusion, the Miró system provides a convenient visual language for specifying security properties. Our future work will concentrate on applying the Miró languages to domains other than security.

A. Nico Habermann
Carnegie Mellon University
School of Computer Science
(412) 268 - 2592
anh@cs.cmu.edu
Graduate Research on Miró and Avalon
Student Reporting - Mark Walter Maimone
N00014 - 88 - K - 0641
1 Oct 88 - 30 Sep 89

4. Miró Publications, Presentations and Reports

References

[1] Mark W. Maimone, J. D. Tygar, and Jeannette M. Wing. Miró semantics for security. In *Proceedings of the 1988 IEEE Workshop on Visual Languages*, pages 45–51, October 1988.

[2] Allen Heydon, Mark Maimone, Amy Moormann, J. D. Tygar, and Jeannette Wing. *Constraining Pictures with Pictures.* Technical Report CMU-CS-88-185, Carnegie Mellon University, School of Computer Science, 1988.

[3] Allan Heydon, Mark W. Maimone, J. D. Tygar, Jeannette Wing, and Amy Moormann Zaremski. Constraining pictures with pictures. In *11th IFIP World Computer Conference*, August 1989.

[4] Allan Heydon, Mark W. Maimone, J. D. Tygar, Jeannette Wing, and Amy Moormann Zaremski. *Miró Tools.* Technical Report CMU-CS-89-159, Carnegie Mellon University, School of Computer Science, July 1989.

[5] Allan Heydon, Mark W. Maimone, J. D. Tygar, Jeannette Wing, and Amy Moormann Zaremski. Miró tools. In *Proceedings of the 1989 IEEE Workshop on Visual Languages*, Rome, Italy, October 1989.

[6] Mark W. Maimone, J. D. Tygar, and Jeannette M. Wing. *Visual Languages and Visual Programming*, chapter Formal Semantics for Visual Specification of Security. Plenum Publishing Corporation, Winter 1989.

A. Nico Habermann
Carnegie Mellon University
School of Computer Science
(412) 268 - 2592
anh@cs.cmu.edu
Graduate Research on Miró and Avalon
Student Reporting - Mark Walter Maimone
N00014 - 88 - K - 0641
1 Oct 88 - 30 Sep 89

5. Miró Research Transitions and DoD Interactions

Through workshop and conference presentations, other researchers are becoming interested in our work. Since the tools are still in the development phase, however, no substantial transitions have occurred.

A. Nico Habermann
Carnegie Mellon University
School of Computer Science
(412) 268 - 2592
anh@cs.cmu.edu
Graduate Research on Miró and Avalon
Student Reporting - Mark Walter Maimone
N00014 - 88 - K - 0641
1 Oct 88 - 30 Sep 89

6. Miró Software and Hardware Prototypes

We have entered into the software development phase of our research. We now have a working editor, printing tool and ambiguity checker. These are all prototype versions, however. Over the next year we will refine these tools, and hope to complete several others. See the paper *Miró Tools* for more details on the status of our software.

Nico Habermann
Carnegie Mellon University
School of Computer Science
(412) 268 - 2592
anh@cs.cmu.edu
ONR Funding for Miró and Avalon
Student Reporting - Stewart Michael Clamen
N00014 - 88 - K - 0641
1 Oct 88 - 30 Sep 89

1 Productivity Measures

Refereed papers submitted but not yet published: 1
Refereed papers published: 0
Unrefereed reports and articles: 2
Books or parts thereof submitted but not yet published: 1
Books or parts thereof published: 0
Patents filed but not yet granted: 0
Patents granted: 0
Invited presentations: 0
Honors received (fellowships, technical society appointments, conference committee role, editorship, etc): 0
Prizes or awards received: 0
Promotions obtained: 0
Graduate students supported >= 25% of full time: 1
Post-docs supported >= 25% of full time: 0
Minorities supported: 0

Nico Habermann
Carnegie Mellon University
School of Computer Science
(412) 268 - 2592
anh@cs.cmu.edu
ONR Funding for Miró and Avalon
Student Reporting - Stewart Michael Clamen
N00014 - 88 - K - 0641
1 Oct 88 - 30 Sep 89

2 Summary of Technical Progress

In the Fall and Winter, we completed work on Avalon/C++, a fault-tolerant, distributed programming language. In the spring, we mounted a design effort towards implementing Avalon/Common Lisp, an extension to Common Lisp with support for reliable, distributed computing.

Over the summer, the Avalon/Common Lisp group designed and implemented a prototype version of our system using CMU Common Lisp, Mach (for remote data transmission), and Camelot (for fault tolerance and primary recoverability). Future work will entail refining our ideas concerning system semantics, and modifying the implementation to conform to them.

Nico Habermann
Carnegie Mellon University
School of Computer Science
(412) 268 - 2592
anh@cs.cmu.edu
ONR Funding for Miró and Avalon
Student Reporting - Stewart Michael Clamen
N00014 - 88 - K - 0641
1 Oct 88 - 30 Sep 89

3 Detailed Summary of Technical Results

Project Goals

A distributed system consists of multiple computers (called sites) that communicate through a network. Distributed systems are typically subject to site crashes and communication link failures. A crash renders a site's data temporarily or permanently inaccessible, while a communication link failure causes messages to be lost. A failure is detected when a site that has sent a message fails to receive a response after a certain duration. The absence of a response may indicate that the original message was lost, that the reply was lost, that the recipient has crashed, or simply that the recipient is slow to respond.

The primary goal of the Avalon Project is to create a set of linguistic constructs designed to give programmers explicit control over transaction-based processing of atomic objects for fault-tolerant applications. These constructs have been implemented as extensions to C++ and Common Lisp. The constructs include new encapsulation and abstraction mechanisms, as well as support for concurrency and recovery. The decision to extend an existing language rather than to invent a new one was based on pragmatic considerations. We felt we could focus more effectively on the new and interesting issues of reliability and concurrency if we did not have to redesign or reimplement basic language features, and we felt that building on top of a widely-used and widely-available language would facilitate the use of Avalon outside our own research group.

Past Accomplishments

Last fall and winter, we completed our efforts on Avalon/C++, an extension to C++.[8] My time was spent debugging the data transmission package (which I designed and built the previous summer), and implementing an example of a highly-available and recoverable data type in our Avalon/C++ system. The data type, a simple counter, grounded at zero, illustrates a number of the unique features of Avalon/C++.

A detailed presentation of the language and its features can be found in [1] and in the part of [3] devoted to discussion of the Avalon/C++ project.

By late winter, our research efforts in Avalon/C++ gave way to the initial design of Avalon/Common Lisp, i.e. a Common Lisp[7] with support for distribution, concurrency, reliability and fault-tolerance. While the goals were similar to those of the Avalon/C++ effort, our design resulted in a different computation model. These differences were the result of influence from a number of directions, both technical and practical.

One of the initial design goals of the Avalon Project was to extend existing languages, rather than invent new ones. In the process of such an extension, it is important to introduce as few new features as possible, and design those features to combine well with the base language's idioms and model of computation. In Avalon/C++, we chose an object-oriented design, consistent with the C++ model. In Avalon/Common Lisp, we strove to extend the language in a similarly consistent manner.

The most significant enhancement we made to Common Lisp was the addition of a new first-class data type, the **evaluator**. An evaluator represents an additional, non-local, Common Lisp evaluator, on which the user can evaluate expressions, install procedures, and modify accessible data. Evaluators are used via two new macros, `remote` and `local`, which direct the thread of computation to a different evaluator.

Avalon/Common Lisp is built on top of three locally-developed systems, CMU Common Lisp, Mach, and Camelot, and runs on IBM-PC/RTs. CMU Common Lisp is one of the first implementations of Common Lisp, and was chosen over other Common Lisp implementations for two reasons. Firstly, it is the only available Common Lisp that runs on the computers the Avalon Group had already available. We also favored the presence of the support and maintenance the locally-managed system provides.

Mach[1], a Unix-like operating system with support for distributed computation, is used to provide communication among the various Avalon processes, and to support process-level concurrency. Camelot[5,3], a machine-independent, high-performance, distributed transaction facility, is used to support the fault-tolerance and reliability we desire.

Over the summer, we designed and built a prototype version of Avalon/Common Lisp. The implementation supports most of the features we had envisioned for the system, with the exception of some of the more expensive features. Please refer to the enclosed Carnegie Mellon Technical Report [2] for details.

My own work has focused on the design of the remote evaluation model for Avalon/Common Lisp, and on the interface between the Camelot recoverable virtual memory system and the Common Lisp data-type system.

The remote evaluation model provides an interesting model of computation for distributed computing. Instead of remote servers and remote procedure calls, the distribution of the computation is generalized to a set of remotely situated Common Lisp **evaluators**, which communicate via "remote evaluator" calls. While not significantly

more computationally expensive from the point of view of the implementation, this model provides a richer environment for the programmer. (See [2] for examples.)

The remote evaluation model was inspired by Stamos' PhD work [6] and by my experiences with Avalon/C++ in previous years. (See last year's Fiscal Report for details of my work with Avalon/C++ .)

The design and implementation of the interface between the Camelot recoverable virtual memory system and Common Lisp mostly involved the integration of the Camelot/Common Lisp interface provided to us by the Camelot Group, and the demands of the standard Common Lisp data types. Camelot, having been written in C, understand only basic C data types. Any useful Lisp interface, however, must understand the more complex data types present there (lists, vectors, structure, etc.). I expended much effort implementing most of the Common Lisp data types on top of the primitive data types provided by the Camelot interface.

Future Goals

My research plans for the next year are twofold. First, during the Fall, I plan to augment the existing Avalon/Common Lisp prototype to support some of the more interesting features of my initial remote evaluation design. In our effort to get the prototype running by the end of the summer, we passed over some of the features that required additional research and extensive implementation efforts, mainly sharing of data among transmitted objects, and propogation of side-effects across evaluator boundaries.

In the new year, we plan to start work on the next generation of Avalon/Common Lisp. At this time, it appears as if we will choose a different Lisp dialect (probably Scheme) as our base language. While Common Lisp provided a fertile testbed for our work, a simpler, cleaner language will allow us to better explore the more formal aspects of our research.

On another front, we have begun discussing our ideas with the ML language[4] community here at SCS, in an effort to understand what a distributed ML might look like.

I personally plan to explore in detail the idea of extending our ideas about distributed computation models with the ML researchers. During our talks, we have discovered that we can learn about our research by trying to cast it into a framework that other language researchers can understand.

References

[1] Accetta, M., Baron, R. V., Bolosky, W., Golub, D. B., Rashid, R. F., Tevanian, Jr., A., and Young, M. W. *Mach: A New Kernel Foundation for UNIX Development*. In: **Proceedings of Summer Usenix**. 1986.

[2] Clamen, S., Leibengood, L., Nettles, S., and Wing, J. *Reliable Distributed Computing with Avalon/Common Lisp*. No. CMU-CS-89-186, Carnegie Mellon University, September 1989. *Submitted to 1990 IEEE Computer Society International Conference; an extended abstract appears as "An overview of Avalon/Common Lisp," in the Proceedings of the Third Workshop on Large Grained Parallel Programming (Pittsburgh, PA, October 10-11, 1989).*

[3] Eppinger, J. L., Mummert, L. B., and Spector, A. Z. **Guide to the Camelot Distributed Transaction Facility including the Avalon Language**. Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[4] Milner, R. *The Standard ML Core Language*. **Polymorphism**, vol. II (1985). *Also Technical Report ECS-LFCS-86-2, University of Edinburgh, Edinburgh, Scotland, March 1986.*

[5] Spector, A. Z., Bloch, J. J., Daniels, D. S., Draves, R. P., Duchamp, D., Eppinger, J. L., Menees, S. G., and Thompson, D. S. *The Camelot Project*. **Database Engineering**, vol. 9 (1986). *Also available as Technical Report CMU-CS-86-166, Carnegie Mellon University, November 1986.*

[6] Stamos, J. W. *Remote Evaluation*. No. MIT/LCS/TR-354, Massachusetts Institute of Techonology, January 1986. *Technical report form of PhD work.*

[7] Steele, Jr., G. L. **Common Lisp: The Language**. Digital Press, 1984.

[8] Stroustrup, B. **The C++ Programming Language**. Addison-Wesley, Reading, Massachusetts, 1986.

[9] Wing, et al., J. *The Avalon Language*. In: **Guide to the Camelot Distributed Transaction Facility including the Avalon Language**, edited by J. L. Eppinger, L. B.Mummert, and A. Z. Spector. Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[10] Wing, et al., J. *The Avalon/C++ Programming Language (Version 0)*. No. CMU-CS-88-209, Carnegie Mellon University, December 1988.

Nico Habermann
Carnegie Mellon University
School of Computer Science
(412) 268 - 2592
anh@cs.cmu.edu
ONR Funding for Miró and Avalon
Student Reporting - Stewart Michael Clamen
N00014 - 88 - K - 0641
1 Oct 88 - 30 Sep 89

4 Publications, Presentations and Reports

Below are the list of Avalon publications I have been involved with during the past year:

References

[1] Jeannette Wing, et al. *The Avalon/C++ Programming Language (Version 0).* Technical Report CMU-CS-88-209, Carnegie Mellon University, December 1988.

[2] S.M. Clamen, L.D. Leibengood, S.M. Nettles, and J.M. Wing. *Reliable Distributed Computing with Avalon/Common Lisp.* Technical Report CMU-CS-89-186, Carnegie Mellon University, September 1989. Submitted to 1990 IEEE Computer Society International Conference; an extended abstract appears as "An overview of Avalon/Common Lisp," in the Proceedings of the Third Workshop on Large Grained Parallel Programming (Pittsburgh, PA, October 10-11, 1989).

[3] Jeannette Wing, et al. The Avalon language. In Jeffrey L. Eppinger, Lily B.Mummert, and Alfred Z. Spector, editors, *Guide to the Camelot Distributed Transaction Facility including the Avalon Language*, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

Nico Habermann
Carnegie Mellon University
School of Computer Science
(412) 268 - 2592
anh@cs.cmu.edu
ONR Funding for Miró and Avalon
Student Reporting - Stewart Michael Clamen
N00014 - 88 - K - 0641
1 Oct 88 - 30 Sep 89

5 Research Transitions and DoD Interactions

A number of researchers, in both academia and industry, have expressed an interest in our Avalon/C++ work. Commercial sites include Microsoft, Texas Instruments, Hewlett Packard, and NCR. Academic sites include Boston Univeristy, University of Massachusetts – Amherst, Concordia University (Montreal), and University of New South Wales (Australia). (A more detailed list is available upon request.)

Our research with Common Lisp has not had much publicity as yet, so not much interaction on that topic has yet occured. Our upcoming publication should remedy that situation.

Nico Habermann
Carnegie Mellon University
School of Computer Science
(412) 268 - 2592
anh@cs.cmu.edu
ONR Funding for Miró and Avalon
Student Reporting - Stewart Michael Clamen
N00014 - 88 - K - 0641
1 Oct 88 - 30 Sep 89

6 Software and Hardware Prototypes

Avalon/C++ is stable and available for distribution. Avalon/Common Lisp is not quite stable, but efforts within the next few months will improve its condition. Plans over the next year are to redesign and reimplement the Avalon/Common Lisp on a different testbed and to design and build a new system to achieve Camelot's without the unnecessary overhead Camelot currently possesses.